27 | 产品测试：除了性能跑分，还能测个啥?

2020-10-12 王磊

分布式数据库30讲 进入课程 >

讲述：王磊

时长 17:43 大小 16.23M

你好，我是王磊，你也可以叫我 Ivan。

这一讲我们的关键词是“测试”。无论是作为程序员还是架构师，我们都不会忽视测试的重要性，它贯穿于软件工程的整个生命周期，是软件质量的重要保障手段。

不过，提到分布式数据库的测试，你也许会有些疑问，我又不是数据库研发人员，还要关心测试吗?

当然是需要了。比如，拿我来说，一名银行的科技人员。银行和很多传统企业一样，多应用系统都是构建在商业软件之上，对于基础软件研发的投入比较有限，所以多数银行是不具备自研分布式数据库能力的。但是，分布式数据库的高并发、高可用性特点，意味着

使用它的一定是非常重要和关键的业务系统。那么，为了保证系统的安全运行，即使不是开发者，我们也仍然需要做大量的验证和测试工作。

说到这，我猜你会想到一个词。对，就是 POC（Proof of Concept）。POC 的意思是概念验证，通常是指对客户具体应用的验证性测试。那验证性测试又具体要测些什么呢？对于数据密集型系统，很多企业的 POC 都会使用 TPC 基准测试。

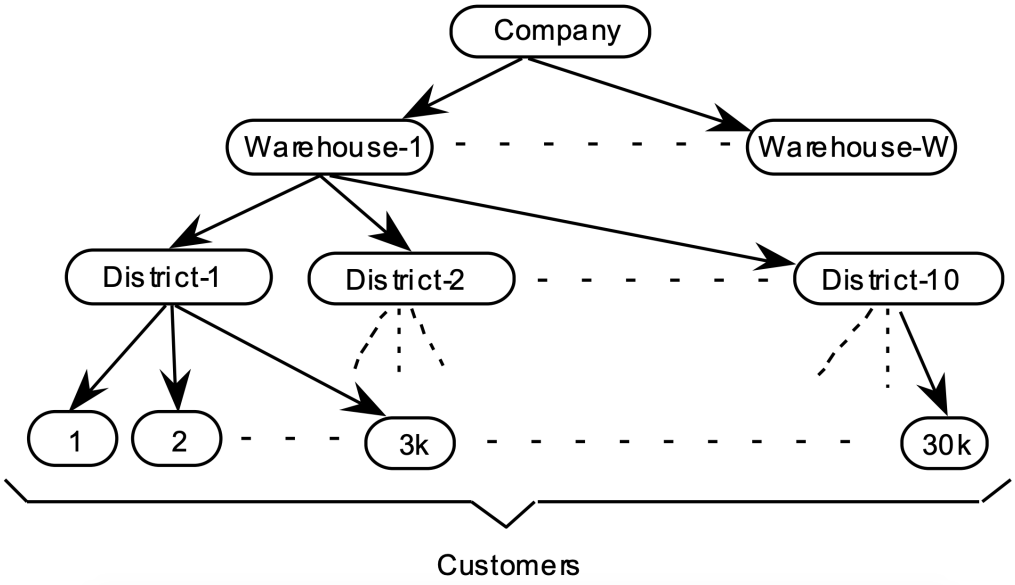
TPC-C

TPC（Transaction Processing Performance Council），也就是国际事务性能委员会，是数十家会员公司参与的非盈利组织。它针对数据库不同的使用场景组织发布了多项测试标准，其中被业界广泛接受有 TPC-C 、TPC-H 和 TPC-DS。

这三个测试标准针对不同的细分场景。简单来说，TPC-C 针对 OLTP 场景；TPC-H 针对 OLAP 场景；而更晚些时候推出的 TPC-DS，在 TPC-H 的基础上又针对数据仓库的建模特点做了更新，并且在 2.0 版本中又增加对大数据技术的针对性测试。

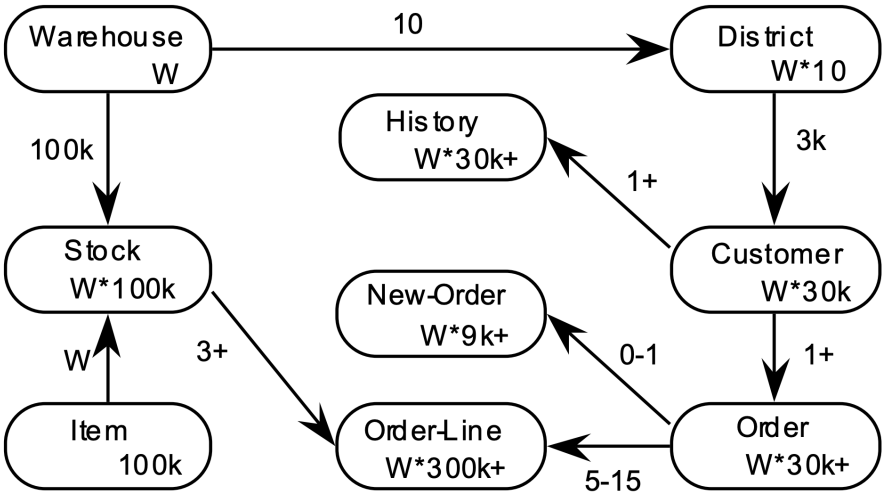
这里，因为我们讲的分布式数据库主要服务于 OLTP 场景，所以我们重点关注 TPC-C。

TPC-C 发布的标准规范中，模拟了一家大型电子商务网站的日常业务。根据规范中的背景设定，这家公司的业务覆盖了很大的地理范围，所以设立了很多的仓库来支持邻近的销售区域，每个仓库都要维护 100,000 种商品的库存记录并支持 10 个销售区域，每个销售区域服务 3,000 个客户。



引自 TPC-C 标准规范（Revision 5.11）

这个场景对应的数据库模型一共包含 9 张表，覆盖了订单创建、支付、订单状态查询、发货和检查库存等五种事务操作。客户会查询已经存在订单的状态或者下一个新的订单。平均每个订单有 10 个订单行（Order-Line），有 1% 订单行的商品在其对应的仓库中没有存货，必须由其他区域的仓库来供货。



引自 TPC-C 标准规范（Revision 5.11）

可以看出，TPC-C 模拟的整个业务场景和我们日常使用的电子商务网站是非常相似的。所以说，TPC-C 测试场景是很有代表性的 OLTP 业务。

TPC-C 为数据库测试提供了一个开放的测试标准，很多 POC 甚至会直接套用这些数据模型和事务操作。可 POC 做起来真有这么容易吗？

如果你实际组织过 POC，就肯定听到过类似这样的一些说法：

“A 公司的数据库是针对 TPC-C 做了优化的，只是测试分数高，实际用起来不行。”

“B 公司的数据库为某个查询语句设置了缓存，这样的测试对我们不公平。”

总之，就是友商为刷高分做了优化，有作弊的嫌疑。

这时，作为组织者该怎么处理呢？可以用一些管理上的办法去协调，但真正解决问题的只有一个，就是要对产品架构有深入的了解，这样才能判断特定的优化措施在自己的真实业务场景下是否普遍有效。如果是普适的，那自然就没问题。

你看，要做好甲方工程师，也是有要求的。

TPC-C 的测试用例除了性能测试，也包含了事务一致性的测试，但实际测试中这部分往往会被忽略。这一方面是为了简化测试过程，另一方面是因为大家会觉得没有必要。既然这些产品都有不少实际案例了，那事务一致性应该就没问题了吧。

可是，对于分布式数据库来说，这真不是个简单的事情，甚至要更加严谨的技术手段来证明。那么，事务一致性方面有没有比 TPC-C 更权威的测试标准呢？

当然有了，这就是 Jepsen。

Jepsen

这个名字是不是有点耳熟？其实我们在第 3 讲介绍事务一致性时就提到过它。Jepsen 是一个开源的分布式一致性验证框架，专门用来测试分布式存储系统，比如分布式数据库、分布式键值系统和分布式消息队列等等。

Jepsen 曾经对很多知名的分布式存储系统进行了测试，而且往往都会发现一些问题，其中分布式数据库就包括 CockroachDB、YugabyteDB、TiDB、VoltDB 和 FaunaDB 等。以下是摘自 Jepsen 官网的所有测试系统列表。

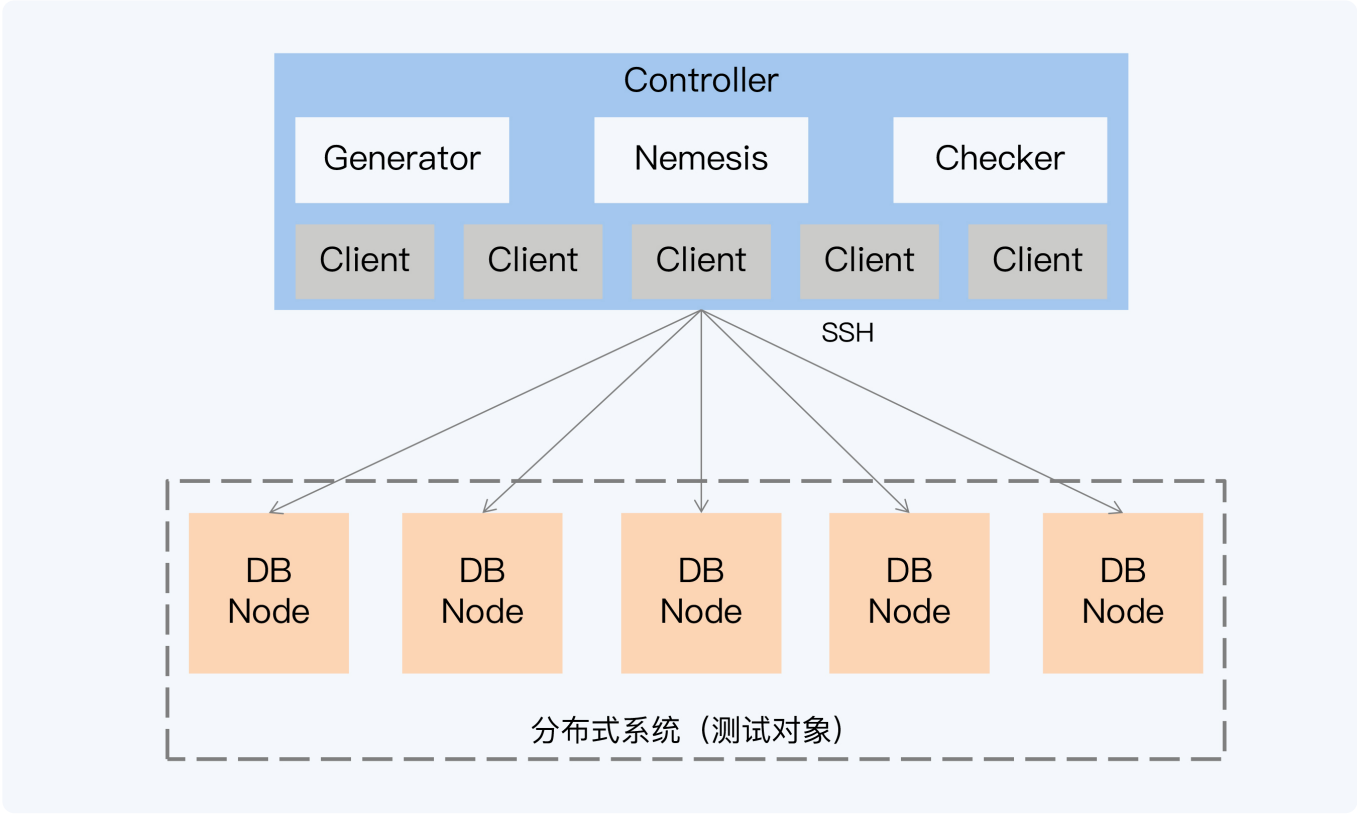
Aerospike	2015-05-04	3.5.4
	2018-03-07	3.99.0.3
Cassandra	2013-09-24	2.0.0
Chronos	2015-08-10	2.4.0
CockroachDB	2017-02-16	beta-20160829
Crate	2016-06-28	0.54.9
Dgraph	2018-08-23	1.0.2
	2020-04-30	1.1.1
Elasticsearch	2014-06-15	1.1.0
	2015-04-27	1.5.0
etcd	2014-06-09	0.4.1
	2020-01-30	3.4.3
FaunaDB	2019-03-05	2.5.4
Hazelcast	2017-10-06	3.8.3
Kafka	2013-09-24	0.8 beta
MariaDB Galera	2015-09-01	10.0
MongoDB	2013-05-18	2.4.3
	2015-04-20	2.6.7
	2017-02-07	3.4.0 - rc3
	2018-10-23	3.6.4
	2020-05-15	4.2.6
NuoDB	2013-09-23	1.2
Percona XtraDB Cluster	2015-09-04	5.6.25
PostgreSQL	2020-06-12	12.3
RabbitMQ	2014-06-06	3.3.0
Redis	2013-05-18	2.6.13
	2013-12-10	WAIT
Redis-Raft	2020-06-23	1b3fbf6
RethinkDB	2016-01-04	2.1.5
	2016-01-22	2.2.3
Riak	2013-05-19	1.2.1
Tendermint	2017-09-05	0.10.2
TiDB	2019-06-12	2.1.7
VoltDB	2016-07-12	6.3
YugaByte DB	2019-03-26	1.1.9
	2019-09-05	1.3.1
Zookeeper	2013-09-23	3.4.5

引自Jepsen官网

要知道这些测试并不是 Jepsen 单方面开展的，而都是和产品团队共同协作完成的，并且这些产品厂商还要向 Jepsen 支付费用。由此可见，Jepsen 在分布式系统测试方面，已经具有一定的权威性。

作为开源软件，你可以从 Github 上下载到Jepsen 的源码，所以有些厂商就在它的基础上定制自己的测试系统。但是比较遗憾的是，Jepsen 的作者选择了一种小众的开发语言 Clojure。我猜，这给多数程序员带来了障碍，因为我能想到的 Clojure 项目似乎只有 Storm。

这里，我们简单介绍下 Jepsen 的架构。



按照 Jepsen 的推荐方案，被测试的分布式系统通常部署在 5 个节点上，而 Jepsen 的程序主要部署在另外的控制节点上。这个控制节点会初始化若干个进程作为分布式系统的访问客户端，当然这里也包括了分布式系统提供的客户端代码。

测试过程中，控制节点要完成三项工作，第一是通过 Generator 生成每个客户端的操作，第二是通过 Nemesis 实现故障注入，最后使用 Checker 分析每个客户端的操作记录来验证一致性。

在整个测试框架中，Nemesis 是特别重要的部分，这是因为 Jepsen 的核心逻辑就是要在各种错误情况下，检测分布式系统还能否正常运行。

“故障注入”在普通测试中并不常见，这里的故障是特指网络分区、时钟不同步这样的底层基础设施层面的问题。因为分布式系统的架构复杂，节点间有千丝万缕的联系，任何软硬件基础设施的错误都可能造成不可收拾的后果，但业务逻辑层面的测试用例又无法覆盖这类场景，所以要靠 Jepsen 来填补这块空白。

说到这，你或许会问，既然故障注入这么重要，那么 Jepsen 注入的这些故障就够了吗？我们是不是要按照自己的业务场景增加一些故障呢？

嗯，不少人也有类似的想法。这就要说一下混沌工程的概念了。

混沌工程

混沌工程（Chaos Engineering）最早是由 Netflix 工程师提出来的。他们给出了这样的定义：混沌工程是在分布式系统上进行实验的学科，旨在提升系统的容错性，建立对系统抵御生产环境中发生不可预知问题的信心。

我们可以从三个层面来理解这个定义。

1. 复杂性

首先，分布式系统的复杂性是混沌工程产生的基础。相比传统的单体系统，分布式系统中包含更多硬件设备，多样化的服务和复杂交互机制。这些因素单独来看似乎是可控的，也有完备的异常处置手段，但当它们组合在一起就会相互影响从而引发不可预知的结果，导致故障发生。而人力是不可能完全阻止这些故障。

混沌工程就是在这些故障发生前，尽可能多的识别出导致这些异常的因素，主动找出系统脆弱环节的方法学。

2. 实验

第二关键点是实验。混沌工程与单纯的故障注入是有区别的，混沌工程的输入是尝试性，目的是探索更多可能发生的奇怪场景，促使正常情况下不可预测的事情发生，从而确认系统的稳定性。我想，正是因为结果具有很大的不确定性，这个过程才会称为“实验”。

最早的混沌测试工具是 Netflix 的 Chaos Monkey，它只会注入一种混乱，那就是随机杀死节点。后来逐步发展，混沌测试框架引入的故障越来越多，包括模拟网络通讯延迟、磁盘故障、CPU 负载过高等等。而混沌测试的观察对象也不仅是一致性（像 Jepsen 那样），而是从系统的各个维度上定义一系列稳态指标，观察混乱注入后系统是否能够快速恢复。

3. 生产环境

第三点，也是混沌工程非常核心的理念，混沌实验在生产环境进行才会获得更大的价值，因为这样才能真正建立起信心，相信系统能抵御各种故障。不过，这个理念也有一定的争议。比如，你今天坐飞机出差，这时混沌工程师要在飞机的控制系统上注入一些故障，想看看系统会不会崩溃，你能接受吗？我想，正常人都会拒绝吧。

显然，对真实业务造成什么后果是做出判断的依据。虽然混沌工程还有一个爆炸半径理念，要限定对生产环境的影响。但是，在生产环境注入混乱来验证系统稳定性的这个理念，对哪些行业适用，进一步又对哪些业务适用，我觉得还是有待探讨的话题。

目前，一些分布式数据库也应用混沌工程进行系统测试，例如 GoldenDB、CockroachDB 和 TiDB。

到这里，对于测试这个话题，我们已经谈了很多，但其实还漏掉了很重要的一点。你能猜到是什么吗？别着急，让我先给你讲一个小故事。

TLA

在前面的课程中，我曾提到过我设计的一款软件 Pharos。它有一个试验特性是在写入数据时，始终保持索引与数据的事务一致性，要知道它的底层是 HBase，本身是不支持跨行事务的，所以说实现这个特性还是有点难度的。

有一次在介绍 Pharos 时，一位同学问我，怎么证明 Pharos 实现了事务一致性呢？我列举了做过的很多破坏性测试，比如杀掉进程、直接重启服务器等等，这些都没影响到事务一致性。但是，讲完之后，我们两个人似乎都对这个答案不太满意。

后来，我就想，我的测试方法还能改进吗？再增加一些异常场景？可似乎都没有本质上的变化。你看，无论 TPC-C、Jepsen 还是混沌工程，虽然方法、理念各不相同，但是都有一个共同点，就是它们只能发现错误，却无法证明正确性。

换句话说，测试是在用证伪的方式来检查软件质量，但是对一个复杂系统来说，测试用例是无法穷尽的，那也就永远不能排除存在 Bug 的可能性。这个结论很让人沮丧，那么，有没有“证明”的方法呢？

方法也是有的，叫做形式化验证（Formal Verification），就是用数学方法去证明我们的系统是无 Bug 的，具体就是用数学工具进行定义、开发和验证（Specification,

Development and Verification）。从一个更高阶的视角来看，不论硬件还是软件，归根结底是在解决数学问题。形式化验证的逻辑就是，如果能够按照严格的数学方法描述设计，那么结果的正确性就也是可以被证明。

但是，要把关键设计按照数学的方式表述一遍，这个实现成本就比较高。所以，形式化验证在软件领域并不常见，而是从硬件领域开始普及，比如 Intel 就在芯片设计中就广泛采用形式化方法。而后随着分布式系统的流行，形式化验证被应用的越来越多。

那么，形式化方法如何在软件工程落地呢？方法就是 Leslie Lamport 提出的 TLA（Temporal Logical of Actions，行为时态逻辑），对又是这位大神。TLA 就是使用数理逻辑来描述系统的时序状态，并验证程序的正确性。

1994 年 Lamport 发表了同名论文。1999 年 Lamport 又发表“Specifying Concurrent Systems with TLA+”论文，提出了 TLA+。TLA+ 是一种软件建模语言，再加上配套的模型校验工具 TLC，这样我们就可以像写程序一样编写 TLA，可以运行来验证最终结果的。2002 年 Lamport 又发布了一本完整的 TLA+ 教科书 *Specifying Systems: The TLA+ Language and Tools for Software Engineers*。因为 TLA+ 使用的是数学化的表达方式，对程序员并不友好，所以后来又出现了 PlusCal。它比 TLA+ 更接近于编程语言，写好的代码可以很方便的转换成 TLA+ 并使用 TLA+ 的模型验证。

小结

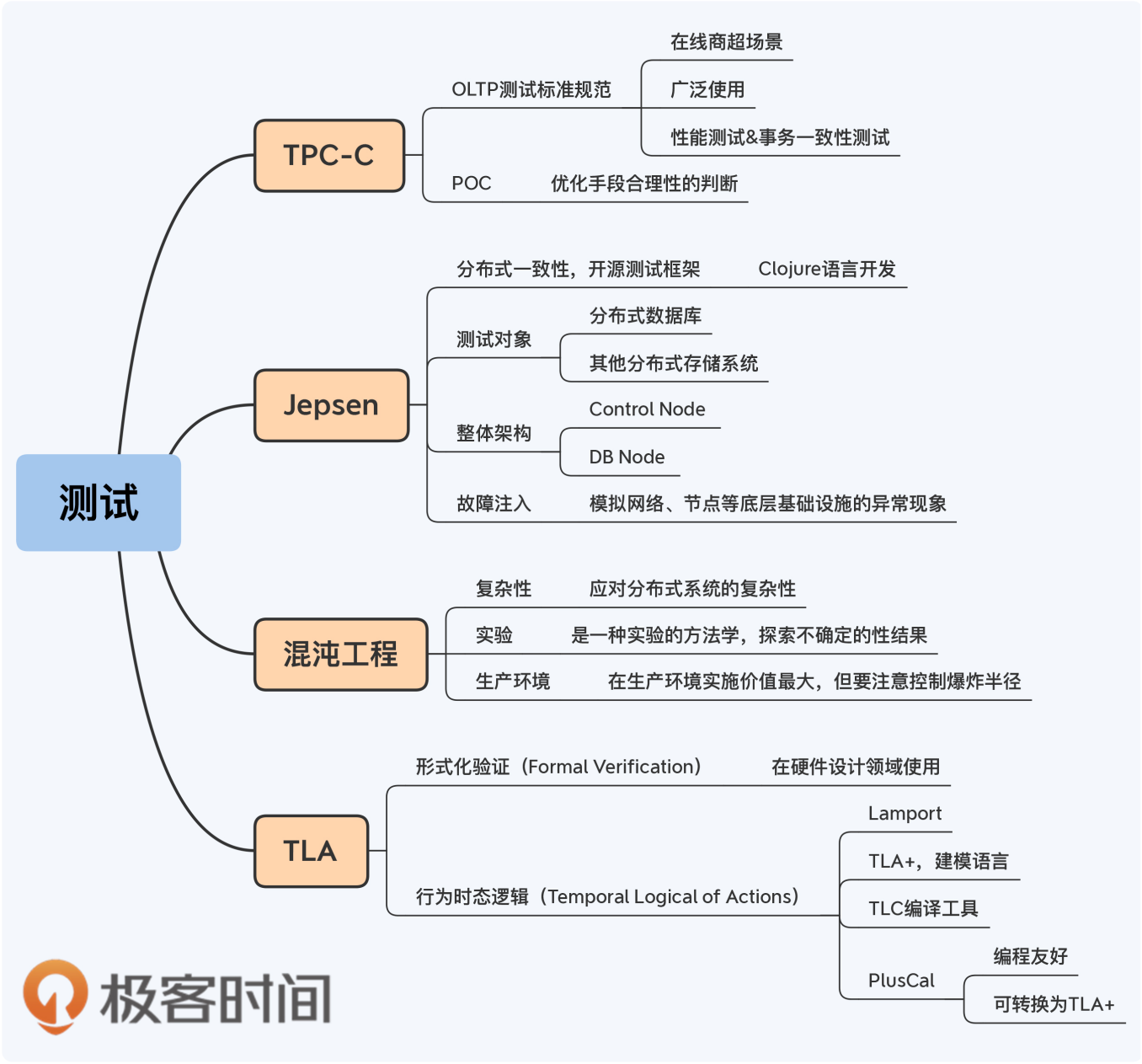
好了，今天的课程就到这里了，让我们梳理一下这一讲的要点。

1. TPC-C 是国际事务性能委员会针对 OLTP 数据库建立的一套测试规范，也是目前广泛接受的测试基准。在很多企业的 POC 测试中会引入 TPC-C，但是由于 TPC-C 的开放性，有的产品会进行针对性优化，使得最终的评测指标失真。要能够分辨优化手段是否对你的业务有普适性，还需要对产品架构的深入掌握。
2. Jepsen 是针对分布式存储系统，进行数据一致性和事务一致性测试的工具。目前已经测试了很多知名系统，具有一定的权威性。Jepsen 通过故障注入的方式进行测试，会覆盖很多在普通测试不能发现的场景。
3. 混沌工程是针对分布式系统提出的方法学。它和测试一样都是为了提高系统的可用性和稳定性，以故障注入为主要手段。混沌工程并不仅是针对某个具体分布式系统提出的。

它强调在生产环境注入故障，在受控的范围内观测系统，体现了反脆弱的思想。总之，混沌工程试图从企业整体运维的视角，用截然不同的理念来提升系统的可用性。

4. 所有的测试方法都只能发现问题，但无法证明正确性。形式化验证可以完美解决这问题，并在很多硬件设计领域有落地实践。Lamport 提出的 TLA 将形式化验证引入软件工程，使用数学工具定义程序逻辑，从而可以达到证明软件无 Bug 的目标。经过不断完善，TLA 从模型到语言、工具建立了一套完备的机制，很多企业也开始使用 TLA 证明关键设计逻辑的正确性。

今天的课程内容可以归结为测试和形式化验证两部分，我们也做了一些比对说明，但并不是说要用验证（Verification）来代替测试（Testing）。通过 TLA 可以验证程序逻辑是否正确，这是个了不起的成就，但是要用数学语言再翻写一遍，付出的成本太高。而且，实际工程中也不可能将所有的代码都转换为 PlusCal 或 TLA+ 来做全面的验证。更多情况下，只是 TLA 来验证关键设计逻辑，剩余的多数代码还是要靠测试来发现问题。总之，测试是不能被验证替代的。



思考题

课程的最后，我们来看下思考题。今天我们的关键词是“测试”，也谈了很多测试方法和理念。不难发现，这些规范和工具都伴随着分布式系统的普及演进。而分布式系统并不限于分布式数据库，其他类型的分布式存储系统也越来越多，所以单一面向数据库的测试工具已经不能满足要求。那么，我今天的问题就是，对于其他类型的分布式存储系统，你知道有哪些主流的测试工具吗？

欢迎你在评论区留言和我一起讨论，我会在答疑篇和你继续讨论这个问题。如果你身边的朋友也对数据库测试这个话题感兴趣，你也可以把今天这一讲分享给他，我们一起讨论。

学习资料

GitHub: *jepsen*

Jepsen: *Analyses*

Leslie Lamport: *A Temporal Logic of Actions*

Leslie Lamport: *Specifying Concurrent Systems with TLA+*

Leslie Lamport: *Specifying Systems: The TLA+ Language and Tools for Software Engineers*

Transaction Processing Performance Council: *TPC BENCHMARK™ C：Standard Specification (Revision 5.11)*

提建议

上一篇 26 | 容器化：分布式数据库要不要上云，你想好了吗？

下一篇 用户故事 | 李兆龙：博观而约取，厚积而薄发

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。